Online Discovery of Similarity Mappings

Alexander Rakhlin RAKHLIN@CS.BERKELEY.EDU
Jacob Abernethy JAKE@CS.BERKELEY.EDU
UC Berkeley, Division of Computer Science

Peter L. Bartlett BARTLETT@CS.BERKELEY.EDU
UC Berkeley, Division of Computer Science and Department of Statistics

Abstract

We consider the problem of choosing, sequentially, a map which assigns elements of a set $\mathcal{A}$ to *a few* elements of a set $\mathcal{B}$. On each round, the algorithm suffers some cost associated with the chosen assignment, and the goal is to minimize the cumulative loss of these choices relative to the best map on the entire sequence. Even though the offline problem of finding the best map is provably hard, we show that there is an equivalent online approximation algorithm, Randomized Map Prediction (RMP), that is efficient and performs nearly as well. While drawing upon results from the "Online Prediction with Expert Advice" setting, we show how RMP can be utilized as an online approach to several standard batch problems. We apply RMP to online clustering as well as online feature selection and, surprisingly, RMP often outperforms the standard batch algorithms on these problems.

1. Introduction

A number of problems in machine learning involve finding constrained similarity mappings or correspondences between two sets. The performance in such problems often depends crucially on the quality of the mapping, feature selection being one example. In this paper, we present an online method for dynamically finding similarities between sets, prove performance guarantees, and apply our method to various settings.

Consider the following general problem. Given sets $\mathcal{A}$ and $\mathcal{B}$, we would like to dynamically find (learn) similarity maps $\pi : \mathcal{A} \to \mathcal{B}$ with the following constraint: $|\text{image}(\pi)| \leq m$ for some given $m \leq |\mathcal{B}|$. In other words, we are looking for a good way to assign elements of $\mathcal{A}$ to a few elements of $\mathcal{B}$, where the quality of the assignment is possibly *changing through time*.

While the unconstrained matching problem can be solved efficiently, the constrained problem is provably hard, as the problem reduces to the `SET COVER` problem. We argue that there are several natural online learning scenarios where the problem of finding a constrained mapping arises, and that the issue of hardness can be avoided by an approximate algorithm, which enjoys provably good performance guarantees.

An important aspect of this constrained mapping problem is the dynamic nature of the cost of a given map. The assumption that the cost of π changes over time is relevant in many real-world applications. This is in contrast to the well-studied machine learning approaches which assume a static nature of the world and the i.i.d. nature of the observed data.

Since the quality of the constrained mappings may change through time, we model our problem as being online: on each round t, we predict a mapping $\pi^t : \mathcal{A} \to \mathcal{B}$ and observe a cost (loss) associated with assigning $a \mapsto b$, for every $a \in \mathcal{A}$ and $b \in \mathcal{B}$, thus receiving feedback as to how good our prediction π^t was. We make no assumption about the nature of this feedback: the cost functions may in fact be adversarially chosen. The objective is to choose a map π^t at every round such that, in the long run, we perform nearly as well as the best fixed map π^* chosen *in hindsight*. This objective fits perfectly with the setting of prediction with expert advice (see Cesa-Bianchi and Lugosi (2006) for a comprehensive treatment).

The problem of discovering a constrained mapping between two sets is quite general and can be applied to a large number of settings often considered in machine

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007. Copyright 2007 by the author(s)/owner(s).

learning. Before diving into details, we mention a few possible applications.

1. **Online Feature Selection:** Assuming that we are given a collection of features $\mathcal{F}$, we can let $\mathcal{A} = \mathcal{B} = \mathcal{F}$. The goal is to choose a representative subset of features at each step, re-evaluating the choices based on the observed data. We therefore consider maps $\mathcal{F} \to \mathcal{F}$ which assign features to the m most representative features. The cost of an assignment $\pi^t : \mathcal{F} \to \mathcal{F}$ is the total dissimilarly between features $\langle f_i \rangle$ and $\langle \pi^t(f_i) \rangle$. The dynamic nature of the problem allows us to adaptively select features and still perform as well as the best choice of features in the long run.

2. **Multitask Feature Selection:** Assume we are given K *related* tasks; for example, the tasks of detecting a face, a vehicle, an animal, etc., in images. We have a number of fixed features F and we would like to find a small number of them which are salient on each of the tasks. Here, we assume that $\mathcal{A} = \{1, \ldots, K\}$ and $\mathcal{B} = F$, or that $\mathcal{B}$ is a collection of small subsets of features F.

3. **Online Data Clustering:** Suppose we are given a fixed number K of non-stationary data points, and we would like find m "prototypes" from these data points such that all the remaining data are similar to one of the prototypes. In other words, we are looking for maps $\pi^t : [K] \to S$ where S is an m-sized subset of $[K]$ and the total distortion, $\sum_{i=1}^{K} d(i, \pi(i))$, is as small as possible. This type of vector quantization can be performed online, where we assume that the data points change with time, or even that each data point is revealed to us slowly, e.g. one feature at a time.

4. **Dynamic Resource Allocation:** There are numerous real-world applications, where a small set of resources has to be adaptively allocated at every time step. As an example, suppose m sensors or light sources have to be turned on or placed in an $N \times N$ network grid so that each of the A moving objects is close to some sensor. The problem corresponds to setting $\mathcal{B}$ to be the N^2 grid points and adaptively choosing a mapping from objects to grid points for sensor placement.

The above examples, as well as other scenarios which fall under the framework presented in this paper, have a flavor of "clustering". One might argue that the problem we present can be solved with batch EM-like clustering methods. While such an approach is reasonable, the contribution of this paper is an efficient method for which we can in fact prove *worst-case performance guarantees*. Moreover, our experiments suggest that the method we propose often performs better than EM.

As we discuss later in the paper, our approach yields vanishing per-round error, while an approach which finds *the best* solution at each step is not only provably hard, but is also inferior in the adversarial setting in terms of the performance guarantees. Indeed, it has been observed by Hannan in 1957 that a prediction strategy has to be randomized – an approach we take.

In the online setting, which better models the real world, we are *required* to make predictions/actions at each round before we see all the data. In fact, our predictions may even influence the environment thus giving us dependent data. This is in contrast to most machine learning algorithms which require an i.i.d. assumption. Our influence on the environment might have a profound positive impact on performance – we might be able to focus on the important aspects of the problem faster, akin to active learning. Surprisingly, we are still able to make a sequence of predictions and suffer a loss comparable to an offline *best* choice[1] – a generally unrealistic choice since the presented data depends on the intermediate actions.

The novel contributions of this paper are the following. We provide a general formulation of online learning of similarity mappings for which an offline version is an NP-hard optimization problem. We prove a theoretical guarantee on the exponentially weighted prediction strategy over the set of mappings which provides an approximate online solution to solve this hard problem. We exhibit an efficient algorithm for implementing this strategy. Finally, we explore applications to feature selection and clustering, with experimental results highlighting the soundness of our approach.

2. Formal Setting

Without loss of generality, suppose $\mathcal{A} = [A] := \{1, \ldots, A\}$, $\mathcal{B} = [B]$ and fix $m \leq B$. Let

$$\Pi_m = \{\pi : \mathcal{A} \to \mathcal{B} \text{ s.t. } |\text{image}(\pi)| \leq m\}$$

be the class of all possible constrained mappings between the two sets. The algorithm we present below competes with the best choice of a mapping from Π_m. Let $c_{i,j}^t \in [0, 1]$ be the cost of assigning $i \mapsto j$ at time $t \in [T]$.

There are two closely related scenarios which we could

[1]In the present paper we do not consider comparators that change through time, although such an analysis would be interesting.

consider in this paper: (a) At each time step we are asked where an individual element i is mapped, and after predicting $i \mapsto j$, the algorithm suffers a loss $c^t_{i,j}$. Alternatively, (b) at each time step we have to predict a complete mapping $\pi^t \in \Pi_m$, and the loss is the sum of the costs of assignments $i \mapsto \pi^t(i)$ for all $i \in \mathcal{A}$.

The present paper is inspired by (Abernethy et al., 2007), who considered scenario (a) in the setting of multitask learning and developed online algorithms with near-optimal performance guarantees. In this paper we consider scenario (b), as it is more natural for the applications we have in mind.

Accordingly, at time $t \in [T]$, we predict a mapping $\pi^t \in \Pi_m$. We then observe the costs $c^t_{i,j} \in [0,1]$ for all $i \in \mathcal{A}$, $j \in \mathcal{B}$ and suffer $\hat{c}^t = \sum_{i=1}^A c^t_{i,\pi^t(i)} \in [0, A]$. The cumulative loss of the algorithm after t steps is $\hat{C}^t = \sum_{s=1}^t \hat{c}^s$. Similarly, for any fixed π, the cost of predicting π at time t is $c^t_\pi = \sum_{i=1}^A c^t_{i,\pi(i)} \in [0, A]$ and the cumulative loss after t steps is

$$C^t_\pi = \sum_{s=1}^t c^s_\pi = \sum_{s=1}^t \sum_{i=1}^A c^s_{i,\pi(i)}.$$

Further, define the unnormalized weights $w^t_{i,j} = \exp\left(-\eta \sum_{s=1}^t c^s_{i,j}\right)$ for some learning rate parameter $\eta > 0$ and set $w^0_{i,j} = 1$ for all $i \in \mathcal{A}, j \in \mathcal{B}$.

We can now formulate our goal as follows: find an algorithm for predicting π^t such that the cumulative loss $\hat{C}^T$ is close to $\min_{\pi \in \Pi_m} C^T_\pi$.

Recall the setting of learning with expert advice. The exponentially weighted prediction strategy (Littlestone & Warmuth, 1994) keeps weights over actions proportional to $\exp(-\eta \cdot \text{Cumulative Loss(action)})$ and randomly draws an action according to the distribution induced by these weights. Corollary 4.2 of (Cesa-Bianchi & Lugosi, 2006) states that such a strategy has, with high probability, a bound on the regret (i.e. the difference between the cumulative loss of the algorithm and the best fixed action) of the order $\sqrt{T \ln N}$, where N is the number of actions.

Our problem of dynamically discovering mappings between sets can be seen as a problem of taking a compound action $\pi^t \in \Pi_m$ at time t. The natural step is thus to employ the results for the exponentially weighted strategy described above. Note that this would require keeping, updating, and sampling from a distribution over Π_m, an approach one might fear is infeasible due to the combinatorial explosion of the size of Π_m. Postponing these computational issues, let us nevertheless derive a performance guarantee for the exponentially weighted strategy over actions Π_m as a consequence of Corollary 4.2 of (Cesa-Bianchi & Lugosi, 2006). Let u^t_π denote the weight of action $\pi \in \Pi_m$ at time t. The strategy requires that

$$u^t_\pi = \frac{\exp\left(-\eta C^t_\pi\right)}{\sum_{\pi' \in \Pi_m} \exp\left(-\eta C^t_{\pi'}\right)}, \tag{1}$$

where C^t_π is the cumulative loss of an action π as defined above. If π^t is chosen randomly according to the distribution induced by weights u^t_π, we obtain the following result.

Theorem 2.1 *Let $\eta = \sqrt{8(m \ln(B/m) + A \ln m)/T}$. Given an algorithm that, on round t, samples a map $\pi \in \Pi_m$ according to the distribution $p(\pi) = u^t_\pi$ described in (1), then with probability at least $1 - \delta$,*

$$\hat{C}^t \le \min_\pi C^T_\pi + A\sqrt{\frac{T}{2}\left(m \ln \frac{B}{m} + A \ln m\right)} + A\sqrt{\frac{T}{2} \ln \frac{1}{\delta}}$$

for any $\delta \in (0,1)$. Here $u^0_\pi = 1$ for all π.

The proof proceeds by observing that the number of different mappings $\pi : \mathcal{A} \to \mathcal{B}$ such that $|\text{image}(\pi)| \le m$ is bounded as $|\Pi_m| \le \binom{B}{m} \cdot m^A$ and, hence, $\ln |\Pi_m| \le m \ln(B/m) + A \ln m$.

The main message of the above theorem is that the per-round regret is diminishing with increasing T, resulting in an $O(1/\sqrt{T})$ approximation to the best offline mapping. Also note that the choice of η depends on the time horizon T, but the standard doubling trick can be employed to remove this requirement (Cesa-Bianchi & Lugosi, 2006).

Of course, the theoretical bound of Theorem 2.1 is useful only if we can demonstrate that the exponentially-weighted scheme over the set of actions Π_m can be efficiently implemented. Recently, there is growing interest in online prediction problems when the decision space is exponentially large yet structured. The problem of this paper is one of such cases, and we now provide an efficient approximate solution.

First, we notice that the linearity of the cost of an action π implies that the weights v_π can be written as

$$\begin{aligned} u^t_\pi &= \frac{\exp\left(-\eta C^t_\pi\right)}{\sum_{\pi' \in \Pi_m} \exp\left(-\eta C^t_{\pi'}\right)} \\ &= \frac{\prod_{i=1}^A \exp\left(-\eta \sum_{s=1}^t c^s_{i,\pi(i)}\right)}{\sum_{\pi' \in \Pi_m} \prod_{i=1}^A \exp\left(-\eta \sum_{s=1}^t c^s_{i,\pi'(i)}\right)} \\ &= \frac{\prod_{i=1}^A w^t_{i,\pi(i)}}{\sum_{\pi' \in \Pi_m} \prod_{i=1}^A w^t_{i,\pi'(i)}}. \end{aligned}$$

Hence, the weight u_π^t of any π can be obtained directly from the $A \times B$ matrix $[w_{i,j}^t]_{i,j}$ by taking a product of weights of individual assignments $i \mapsto \pi(i)$ and normalizing. For now, we put aside the feasibility issues associated with computing the normalization as these are addressed in Section 3. After the costs $c_{i,j}^t$ are observed, the new unnormalized weights of the mappings $i \mapsto j$ are naturally updated as $w_{i,j}^t = w_{i,j}^{t-1} \cdot \exp\left(-\eta c_{i,j}^t\right)$ for all $i \in \mathcal{A}, j \in \mathcal{B}$. This solves the problem of keeping and updating the weights over exponentially many actions Π_m. However, to employ Corollary 4.2 of (Cesa-Bianchi & Lugosi, 2006), we must *sample* actions according to the weights u_π^t. This computationally difficult problem has been addressed in (Abernethy et al., 2007) in the setting of multitask learning. The idea is to use a random walk over subsets of size m, as presented in the next section.

3. Randomized Map Prediction Algorithm

The algorithm we present next keeps a matrix of weights $[w_{i,j}^t]_{i\in\mathcal{A},j\in\mathcal{B}}$, such that $w_{i,j}^t = \exp\left(-\eta \sum_{s=1}^t c_{i,j}^s\right)$ for some $\eta > 0$. Let $\mathcal{S}_m = \{S \subset \mathcal{B} : |S| = m\}$. On round t we sample a map π^t according to the following procedure.

Algorithm 1 Randomized Map Prediction (RMP)

```
1: Input: Round $t$; Parameter $m < B$; Matrix $[w_{i,j}^t]_{i\in\mathcal{A},j\in\mathcal{B}}$
2: Sample $S \in \mathcal{S}_m$ according to
   $$P(S) = \left(\prod_{i=1}^{A}\sum_{j\in S} w_{i,j}^t\right) \Big/ \left(\sum_{S'\in\mathcal{S}_m}\prod_{i=1}^{A}\sum_{j\in S'} w_{i,j}^t\right)$$
3: Order $S = \{j_1, \ldots, j_m\}$ s.t. $j_1 < j_2 < \ldots < j_m$
4: Sample $\phi : [A] \to [m]$ by sampling $\phi(i)$ independently for all $i \in [A]$, $k \in [m]$:
   $$P(\phi(i) = k|S) = \frac{w_{i,j_k}^t}{\sum_{j\in S} w_{i,j}^t}$$
5: Output $\pi^t$ defined by $\pi^t(i) := j_{\phi(i)}$
```

The above procedure for choosing π^t is as follows: first sample a set $S \in \mathcal{S}_m$, which becomes the image(π), then sample the assignment $\pi(i) \in S$ for every $i \in \mathcal{A}$ according to the weights $\{w_{i,j}^t : j \in S\}$. This procedure produces an m-constrained map, and (Abernethy et al., 2007) showed that this sampling induces a distribution resulting in the bound of Theorem 2.1.

Hence, an algorithm which updates the $A \times B$ matrix of weights as $w_{i,j}^t = w_{i,j}^{t-1} \cdot \exp\left(-\eta c_{i,j}^t\right)$ and predicts π^t according to Algorithm 1, enjoys the regret bound of Theorem 2.1. However, we are not done yet, as step 2 of Algorithm 1 still requires computing $\binom{B}{m}$ quantities – a number smaller than the size of Π_m, but still infeasible for large B. The Metropolis-Hastings method below (Algorithm 2) approximates step 2 by performing a random walk over $\mathcal{S}_m$.

Algorithm 2 Sampling an m-subset of $\mathcal{B}$

```
1: Input: Matrix $[w_{i,j}^t]_{i\in\mathcal{A},j\in\mathcal{B}}$; number of rounds $R$
2: Start with some $S_0 \in \mathcal{S}_m$
3: **for** $r = 0$ to $R-1$ **do**
4:     Uniformly at random, choose $s' \in [B] \setminus S_r$ and $s \in S_r$. Let $S'_r = S_r \cup s' \setminus s$.
5:     Calculate $\omega(S_r) = \prod_{i=1}^A \sum_{j\in S_r} w_{i,j}^t$ and $\omega(S'_r) = \prod_{i=1}^A \sum_{j\in S'_r} w_{i,j}^t$
6:     With probability $\min\left\{1, \frac{\omega(S'_r)}{\omega(S_r)}\right\}$,
       set $S_{r+1} \leftarrow S'_r$, otherwise $S_{r+1} \leftarrow S_r$
7: **end for**
8: Output: $S_R$
```

The final issue is the choice of R. While the distribution induced by our random walk will certainly approach the desired distribution as we increase R, it must be shown that our Markov chain mixes *quickly*. Unfortunately, such results are scarce and can only be proven in particular cases. On the other hand, our experimental results – and those reported in (Abernethy et al., 2007) – show that, in all cases we have tested, this Markov chain mixes very rapidly.

4. A General Optimization Problem

One can see that the previous section provides a randomized online algorithm which approximates the following objective function:

$$\Phi_m = \min_{j_1,\ldots,j_m\in[B]} \sum_{i=1}^{A} \min_{k\in\{1,\ldots,m\}} \sum_{t=1}^{T} c_{i,j_k}^t \qquad (2)$$

for an arbitrary sequence of $c_{i,j}^t \in [0,1]$ for all $i \in [A], j \in [B], t \in [T]$. As the next theorem shows, the exact minimization problem is provably hard, while the online approximation to within $O(\sqrt{T})$ is achieved by our algorithm.

Theorem 4.1 *Finding Φ_m is NP-hard. Furthermore, this is true even when $T = 1$ and the $c_{i,j}^t$ are restricted to $\{0,1\}$.*

We now sketch a proof. Suppose we could solve (2) exactly. We claim that then we could solve the `SET COVER` problem. Indeed, without loss of generality let $S = \{1, \ldots, A\}$ and $\mathcal{D} = \{D_1, \ldots D_B\}$ a collection of subsets of S. Construct the binary $A \times B$ matrix G as follows: $G_{i,j} = 0$ if $i \in D_j$ and 1 otherwise. Note that finding a set cover of size m is equivalent to finding a subset $\{j_1, \dots, j_m\}$ of columns (subsets of S) such that for all $i \in [A]$, $G_{i,j} = 0$ for at least one $j \in \{j_1, \dots, j_m\}$, i.e. $\Phi_m = 0$. If we could solve (2) exactly, we would be able to find an m-sized set cover, if it exists.

Hence, finding the best mapping π given a matrix of cumulative costs at time t is hard. More importantly, we note that, even if we could efficiently perform the above optimization, simply predicting with the best expert π at time t, known as "Follow the Leader", is not a good prediction strategy in the adversarial setting. We illustrate this fact in Section 5.1. We conclude that our online algorithm has several virtues: it has performance competitive with the best offline choice, even when finding the best is hard, and it randomizes its prediction at each step to be robust in the adversarial setting.

5. Applications and Experiments

We now show how RMP can be applied to two common machine learning problems: data clustering and feature selection (dimensionality reduction). Since our focus is on the online learning setting, we cast each of these problems within this framework and also compare our online approach to several batch algorithms.

In each of the examples below, we are considering objects $\{\boldsymbol{v}_j\}$ that are typically members of Euclidean space, and we will use the term "vector." However, this is not entirely necessary, as we require only that each object $\boldsymbol{v}_j$ can be described by some sequence $\langle v_j^1, v_j^2, \dots, v_j^T \rangle$, and that we have a distortion function $d(\cdot,\cdot)$ which can be represented as a coordinatewise sum. That is, $d(\boldsymbol{v}_i, \boldsymbol{v}_j) = \sum_{t=1}^T d(v_i^t, v_j^t)$. Here, for convenience, we slightly abuse notation, as we write $d(\cdot,\cdot)$ as both a function on $\boldsymbol{v}_i$ as well as on individual coordinates v_i^t. As an example, if $\boldsymbol{v}_j$'s are in $\mathbb{R}^T$, we can choose the Euclidean distance squared, $d(\boldsymbol{v}_i, \boldsymbol{v}_j) = \sum_{t=1}^T (v_i^t - v_j^t)^2$, or L_1 distance $d(\boldsymbol{v}_i, \boldsymbol{v}_j) = \sum_{t=1}^T |v_i^t - v_j^t|$.

5.1. Online Data Clustering

Consider the following scenario. We have K objects, where object $j \in [K]$ is defined by a T-dimensional vector $\boldsymbol{v}_j = (v_j^1, \dots, v_j^T)$, and we have some distortion function $d(\cdot,\cdot)$ as described above. Assume that we cannot observe the entire vector and, instead, on every round $t = 1, \dots, T$, we receive v_j^t for each object j. In other words, the descriptions of each object are revealed to us slowly. More generally, we could consider a setting where our objects are in fact *moving*, and the position of object i on round t is v_i^t, where v_i^t is itself a position vector.

We are interested in choosing a clustering of the K objects into m groups. One can see that this problem can be interpreted as the problem of finding a map $\pi : [K] \to [K]$ such that $|\text{image}(\pi)| \le m$. We can consider $\text{image}(\pi) = \{i_1, \dots, i_m\}$ to be a set of m or fewer "prototype" objects, and the clusters are exactly the sets $\pi^{-1}(i_1), \dots, \pi^{-1}(i_m)$. If a clustering is chosen *offline*, that is where each vector $\boldsymbol{v}_j$ is fully revealed, then the "cost" of this clustering is simply $\sum_{j=1}^K d(\boldsymbol{v}_j, \boldsymbol{v}_{\pi(j)}) = \sum_{t=1}^T \sum_{j=1}^K d(v_j^t, v_{\pi(j)}^t)$.

The objective function we are trying to minimize for this particular clustering method is the same as the "K-medoids" algorithm (Hastie et al., 2002). This particular objective differs from that of other standard clustering algorithms, e.g. K-means or EM clustering, where the goal is to find m "cluster centers" which themselves need not be members of the dataset but arbitrary points in the ambient space. In general, performing K-medoids clustering is computationally more expensive than K-means (Hastie et al., 2002, page 468).

Now assume that the algorithm is required to cluster the objects while the data is revealed. That is, the algorithm would like to find a dynamic clustering with a cost that is minimal relative to the cost of the best offline clustering. At each round t, the algorithm has, for each i and j, a record of the total distortion $d(\boldsymbol{v}_i^{1:t-1}, \boldsymbol{v}_j^{1:t-1})$. The algorithm must choose a map $\pi^t : [K] \to [K]$ such that $|\text{image}(\pi^t)|$ is a set of m or fewer prototypes. The next coordinates $v_1^t, \dots, v_K^t$ are revealed, and the algorithm suffers $\sum_{i=1}^K d(v_i^t, v_{\pi^t(i)}^t)$. The total loss of the dynamic clustering $(\pi^1, \pi^2, \dots, \pi^T)$ is then $\sum_{t=1}^T \sum_{i=1}^K d\left(v_j^t, v_{\pi^t(j)}^t\right)$. At the end of the sequence, the algorithm's cumulative regret is

$$\sum_{t=1}^T \sum_{i=1}^K d\left(v_i^t, v_{\pi^t(i)}^t\right) - \min_{\pi \in \Pi_m} \sum_{i=1}^K d\left(\boldsymbol{v}_i, \boldsymbol{v}_{\pi(i)}\right).$$

This problem translates directly to our dynamic mapping problem and we thus obtain a natural online algorithm for choosing a clustering. Here, $\mathcal{A} = \mathcal{B} = [K]$, $c_{i,j}^t := d(v_i^t, v_j^t)$, and $C_\pi^t = \sum_{i=1}^K d(\boldsymbol{v}_i^{1:t}, \boldsymbol{v}_{\pi(i)}^{1:t})$. On each round, by utilizing the random walk described in Algorithm 2, we sample a map $\pi^t \in \Pi_m$, and the algorithm returns the clustering π as its prediction.

We experimented with this algorithm by generating a set of data points in high-dimensional space, $\boldsymbol{v}_i \in \mathbf{R}^d$

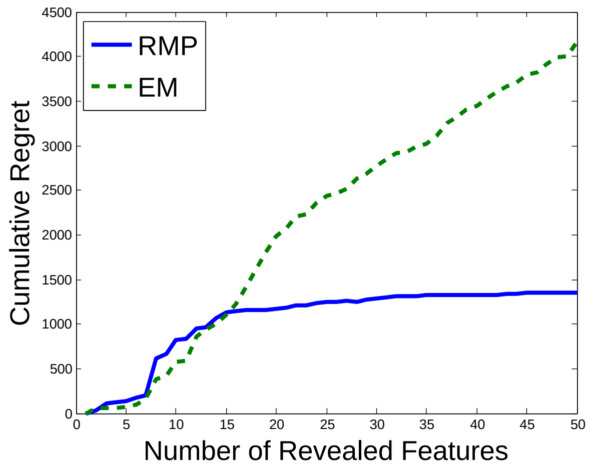

Figure 1. The regret of RMP and EM clustering in the online clustering task.

where $T = d = 50$ and $i = 1, \ldots, 400$. The $\boldsymbol{v}_i$'s were sampled from a mixture of four gaussian distributions with randomly chosen centers. The width of all guassians was chosen so that the gaussians would be easily separable with all 50 dimensions, but non-separable with fewer than (roughly) 15 dimensions. Since we generated the data, we were able to efficiently compute the optimal offline clustering π^*.

In Figure 1 we plot $\hat{C}^t - C^t_{\pi^*}$, the cumulative performance of our dynamic clustering relative to the performance of π^*, for $t = 1, \ldots, 50$. We observe that the regret of our algorithm is sublinear in T and appears to be roughly $O(\sqrt{T})$, as we would expect given our bound of vanishing per-round regret (Theorem 2.1).

In addition, we applied the EM clustering method, adapted for the online framework as follows: on round t we used the EM algorithm to cluster the set of partially revealed vectors, $\{\boldsymbol{v}_j^{1:t-1}\}_j$, and we defined the clustering π to be the map which sends every data point in each cluster to the data point nearest to the centroid of this cluster. The regret of EM was also plotted in Figure 1. The performance was quite variable and here we chose a typical run. EM clustering suffers from problems of local minima, particularly when the data is high-dimensional, and this may be the reason for its relatively poor performance. We include this second graph not to provide a fair comparison with EM but to point out that our sampling algorithm is apparently quite robust. Interestingly, other experiments not reported in this paper suggest that RMP performs substantially better than EM on sparse, high-dimensional data.

In addition to the difficulties mentioned above, there is a more subtle issue associated with applying a "find the best" algorithm, such as K-means. A central theme in online learning is that a learner must always "hedge" his trust in any particular decision, for the best decision historically may not be the best decision today. This is why we *sample* a map π on each round, rather than just choose the π with the lowest loss. Motivated by a similar example in (Kalai & Vempala, 2005), consider dynamically clustering the following three data vectors: $A := \langle .5, 1, 1, 1, 1, \ldots \rangle, B := \langle 0, 1, 0, 1, 0, \ldots \rangle, C := \langle 0, 0, 0, 0, 0, \ldots \rangle$ Assume we are looking for two clusters. After round 1, the optimal clustering is $\{A\}$ and $\{B, C\}$; after 2 rounds, the optimal clustering is $\{A, B\}$ and $\{C\}$; after 3 rounds, it's $\{A\}$ and $\{B, C\}$ again; and so on. Notice, if we always choose historically the best clustering, then on the next round we always suffer *maximal loss*. The total loss of this "follow the leader" style algorithm is going to be at least T after $T+1$ time steps, while the loss of the best fixed clustering will be at most $T/2$. Thus the regret of this approach will be linear in T. On the other hand, the bound in Theorem 2.1 tells us that the regret of our *randomized* algorithm will be $O(\sqrt{T})$, and thus the per-round regret will approach 0 for large T.

5.2. Online Feature Selection

The problem of feature selection, too, can be interpreted as a problem of finding an m-constrained map π. Here, we assume we are given F features, a distortion function $d(\cdot, \cdot)$, and we would like to choose a map $\pi : [F] \to [F]$ which maps our feature set to a small set of features and minimizes distortion. However, in contrast to clustering, where we were mostly interested in the clusters $\pi^{-1}(i_1), \ldots, \pi^{-1}(i_m)$, we are now interested in the actual prototype objects $i_1, \ldots, i_m$ which will be our selected features. Note that, unlike the majority of feature selection algorithms, we obtain these features in an entirely unsupervised fashion. It is thus rather surprising, as we demonstrate below, that we can outperform supervised selection methods.

Feature selection in an online setting was previously studied by (Levi & Ullman, 2006), whose method involves adding and removing features to the set of selected features one-by-one at each round. While their paper develops several good heuristics to choose relevant features, we are concerned in this work with algorithms with provable theoretical guarantees, such as the bound in Theorem 2.1. We also note that our chosen subset of features can completely change between rounds, allowing faster adaptation to new information in a changing environment.

The feature selection problem that we pose here is quite similar to the clustering problem discussed above, and we give a brief outline. Assume we are given a sequence of data points x_t, for $t = 1, 2, \ldots, T$. Assume that each data point is described by a set of F features, $x_t = \langle f_1^t, f_2^t, \ldots, f_F^t \rangle$. After t rounds, we can consider the *feature vector* $\boldsymbol{f}_i^{1:t} = \langle f_i^1, f_i^2, \ldots, f_i^t \rangle$ for the ith feature. We are also given some distortion function $d(\cdot,\cdot)$ on feature vectors with the properties mentioned in the beginning of this section. We define the *online* feature selection problem as follows. On round t, the algorithm has access to the total distortions $d(\boldsymbol{f}_i^{1:t-1}, \boldsymbol{f}_j^{1:t-1})$ for each i and j, then must choose an m-constrained map $\pi^t : [F] \to [F]$, and suffers $\sum_{i=1}^F d\left(f_i^t, f_{\pi^t(i)}^t\right)$. The goal of the algorithm is to perform nearly as well as the best offline map in hindsight, that is, to minimize

$$\sum_{t=1}^{T} \sum_{i=1}^{F} d\left(f_i^t, f_{\pi^t(i)}^t\right) - \min_{\pi \in \Pi_m} \sum_{i=1}^{F} d\left(\boldsymbol{f}_i, \boldsymbol{f}_{\pi(i)}\right). \quad (3)$$

Of course, this way of posing the feature selection problem is somewhat unusual. Often, we are given labelled data and a feature selection method is used to choose a set of m features that are most helpful in predicting the labels. For instance, in analyzing gene expression data to investigate genes associated with a certain type of cancer, the predictive accuracy of a set of genes might be used as a proxy for the biological relevance of those genes; see, for example, (Guyon et al., 2002). In that approach, we should be searching for features that are *informative*, as opposed to representative, and we would say that a feature set S is better than S' if a classifier trained using the features in S performs better than that trained using S'.

On the other hand, the objective (3) has several advantages. First, it is unsupervised, and can thus be applied more generally. Second, the alternative measure, feature "informativeness", is hard to determine when the data are sparse. Third, we can prove bounds on how well our method will perform in the long run. Lastly, by selecting features without labels, we are less prone to overfitting, especially when there is a large number of features but only few data points.

We can apply RMP in the following simple way. Let $\mathcal{A} = \mathcal{B} = [F]$. On round t, we set $c_{i,j}^t := d(f_i^t, f_j^t)$, and thus $C_\pi^t = \sum_{t=1}^T \sum_{i=1}^F d\left(f_i^t, f_{\pi(i)}^t\right)$ for any π. Implementing our random walk, we sample a map $\pi \in \Pi_m$ and choose image(π) as our feature set on this round.

Due to Theorem 2.1, we have a performance guarantee for the dynamically chosen feature set, at least with respect to the measure in (3). Yet it is also interesting to consider how well these features perform when used in classification. We have therefore tested our feature selection on the well known MNIST dataset (LeCun et al., 1998) of handwritten digits. Each digit lies on a 28×28 grid, and each pixel in the grid has an integer grayscale value in the range 0 to 255. Here we consider each data point as a $784 = 28^2$ feature vector, and we note that each method we tested has no access to positional information. The dataset consists of 10 classes, and in our experiments we consider classification between digits '1' and '8', an easy task, and digits '3' and '8', a slightly harder task.

On every round, we provide various algorithms with a new data point, and each algorithm must repeatedly return a set of features. We implemented the following four methods of feature selection: (a) A random set of m features; (b) A set of features chosen by Recursive Feature Elimination (RFE) (Guyon et al., 2002) – a *supervised* feature selection method that recursively trains a kernel classifier and eliminates features based on their weight of the returned classifier [2]; (c) A set of m features chosen using RMP; (d) No feature selection, that is, using all 768 features.

After choosing features, we used ridge regression to build a linear classifier with those feature sets. We report the performance of each selection method in Figure 2. We consider the "1 vs. 8" and the "3 vs. 8" classification problems (which are representative of other two-class classification problems), and we test values $m = 10$ and $m = 50$. The most interesting comparison is between RMP and RFE. RMP doesn't appear to perform well with only a few data points. However, when given at least 50 data points, it clearly outperforms RFE for the larger choice of m. This is particularly surprising since RFE is a supervised selection method, while RMP simply picks features based on their similarities to other features.

To get a feel for what our feature selection algorithm is doing, we display the output of RMP on the 28×28 feature grid in Figure 3. Given 50 random images from the "1 vs. 8" training set, and the "3 vs. 8" training set, we sample a map $\pi \in \Pi_m$, which assigns our pixel set to an m-sized subset of pixels. Given this π, we assign the same color to pixels i and j if $\pi(i) = \pi(j)$. Recall, the algorithm receives no positional information – we simply assign features based on similarities across data points. As we see from the images, RMP finds strongly representative regions in our features space without topological information or image labels.

[2]We applied the built-in Spider implementation of RFE, found at `http://www.kyb.tuebingen.mpg.de/bs/people/spider`

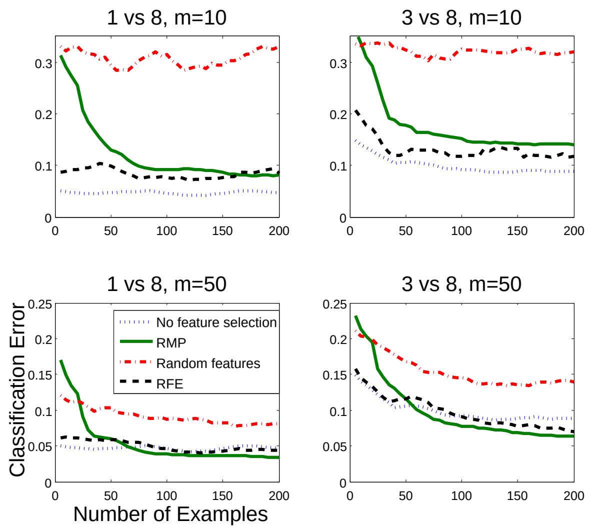

Figure 2. The performance of RMP on the MNIST dataset for different m and two tasks. We compare RMP to RFE and two baseline methods.

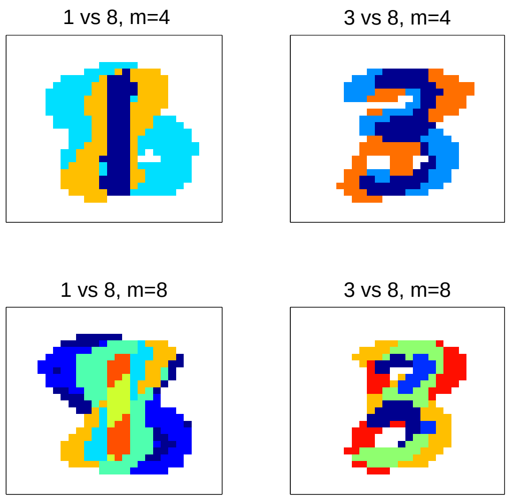

Figure 3. A visualization of the output of RMP on 50 unlabelled images of handwritten digits. The colored regions represent sets of pixels that map to the same pixel under π.

6. Conclusions

In recent years, one witnesses an increasing number of interesting applications of the online experts framework to solving optimization problems. Clever reductions of convex problems as well as provably hard problems to the exponentially weighted prediction strategy (e.g. Arora et al., 2005) yield simple and elegant online approximation algorithms. Structured prediction problems, such as finding shortest paths in graphs (Kalai & Vempala, 2005), can sometimes be solved in the expert prediction framework despite a combinatorial explosion in the number of possible actions.

The present paper makes a connection between the setting of prediction with expert advice and online discovery of similarity mappings between two sets (clustering being one example). The connection allows us to have a theoretical bound on the performance of our algorithm (RMP) for this non-convex optimization problem. We overcome the difficulty of predicting with actions from a very large set by exploiting the structure of the set of mappings Π_m. In particular, we show that only an $A \times B$ matrix of weights is required to keep track of the weights over actions, making our algorithm truly online. An efficient sampling technique for this problem performs well in the experiments, matching the theoretical guarantees.

Acknowledgments

We acknowledge the support of DARPA under contract FA8750-05-2-0249.

References

Abernethy, J., Bartlett, P., & Rakhlin, A. (2007). Multitask learning with expert advice. *COLT 07*.

Arora, S., Hazan, E., & Kale, S. (2005). The multiplicative weights update method: a meta algorithm and applications. Manuscript.

Cesa-Bianchi, N., & Lugosi, G. (2006). *Prediction, learning, and games*. Cambridge University Press.

Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. *Machine Learning*, *46*, 389–422.

Hastie, T., Tibshirani, R., & Friedman, J. (2002). *The elements of statistical learning - data mining, inference, and prediction*. Springer.

Kalai, A., & Vempala, S. (2005). Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, *71*, 291–307.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*, 2278–2324.

Levi, D., & Ullman, S. (2006). Learning to classify by ongoing feature selection. *Proceedings of the The 3rd Canadian Conf. on Computer and Robot Vision*.

Littlestone, N., & Warmuth, M. K. (1994). The weighted majority algorithm. *Inf. Comput.*, *108*, 212–261.